정보기술응용연구
제1권 제3·4호
1999년 12월

암호 알고리즘의 실용적인 키 생성 모델 구현

이 형*, 김 창 영**

요 약

대부분의 암호이론은 공개되어 있기 때문에 정보보안 기술의 안전성은 암호 알고리즘과 키 길이에 의존성이 크다. 본 논문에서는 해쉬함수와 카오스 함수를 이용하여 암·복호화를 위한 권장 키 길이보다 작은 길이의 일회성을 갖는 공개키와 비밀키를 생성하여 공개키 암호 알고리즘의 대표격인 RSA 암호방식에 적용하여 본다. 소인수 분해 알고리즘의 개선·발전과 시스템의 처리속도 증가에서 오는 키길이 증가 문제를 해결하므로 스마트 카드와 같은 제한된 메모리에서 실용적으로 사용할 수 있을 뿐만 아니라, 암·복호화를 수행하는 처리 시간을 단축 시킬 수 있으며, 키 관리면에서도 여러개의 공개키/비밀키를 사용하는 경우보다 실용적이다.

*) 대전대학교 정보통신공학과 교수
**) 대전대학교 정보통신공학과

1. 서 론

초기의 인터넷은 단순한 데이터의 교류였으나, 이제는 전자상거래, 금융거래, 기업업무 등에서 사용되는 실정이다. 인터넷이 개방화되면서 정보의 유출 및 도용, 파괴, 위·변조, 바이러스 유포, 해킹과 같은 인터넷의 역기능 현상이 증가하고, 정보 보안 인프라 구축에서 많은 문제점이 출현되고 있다.

대부분의 암호이론이 공개되어 있기 때문에 정보 보안의 안정성은 암호 알고리즘과 키길이에 의존성이 크다. 비밀키 암호 알고리즘(symmetric cryptography)의 경우 인증문제, 키 관리 문제, 키 분배 문제와 같은 개선점을 수반하고 있지만, 공개키 암호 알고리즘(Asymmetric cryptography)보다 키의 길이가 작으므로 암/복호화를 연산하는 시간이 감소되는 이점을 가지고 있다. 반면에, 공개키 암호 알고리즘은 TCP/IP 프로토콜 기반에서 안전하지 않은 채널 상에 암호화된 데이터의 복호키를 흘려보내지 않는다는 점과 비밀키 암호 알고리즘처럼 키분배 센터(KDC)를 따로 두지 않아도 되는 이점을 가지고 있다. 그러나 인수분해 알고리즘의 발달, 병렬시스템, 양자 컴퓨터 등이 발달함에 따라 키를 계산하는 시간이 짧아지고 있다[1,2].

그 예로, 미국의 암호업체 RSA 데이터 시큐리티가 개최한 암호해독대회에서 97년에 56bit DES로 암호화된 메시지를 96일에 걸쳐 해독해 냈으며, 99년에는 '딥크랙'이란 암호해독 S/W와 인터넷에 연결된 1만 여대의 병렬 컴퓨터를 사용해 22시간 15분만에 해독해냈다.

뿐만 아니라 암/복호화 연산시간은 키길이에 비례하기 때문에 키의 길이가 무한정 길어져서도, 짧아져서도 안된다. [표-1]에서는 이러한 권장키 길이의 변화를 보여준다[1].

[표-1] 공개키 알고리즘의 권장 키길이

Year	vs. Individual	vs. Corporation	vs. Government
1995	768	1280	1536
2000	1024	1280	1536
2005	1280	1536	2048
2010	1280	1536	2048
2015	1536	2048	2048

또한 공개키 알고리즘은 비밀키 알고리즘에 비해 메시지를 암호화하는데 소요되는 시간이 길기 때문에 일반적으로 메시지를 암호화할 때는 비밀키 방식을 사

용하고 이 방식에 사용된 키를 암호화하는데 공개키 방식을 사용한다.
따라서 본 논문에서는 해쉬함수(Hash)와 카오스 함수(Chaos)를 이용하여 불법적인 실체에 의한 키의 재사용을 억제하고, 권장 키 길이보다 작은 길이의 일회성을 갖는 공개키와 비밀키를 생성하여 공개키 암호 알고리즘의 대표 격인 RSA 암호방식에 적용하여 본다. 이러한 일회성 키를 사용함으로써 소인수 분해 알고리즘의 개선과 시스템의 발전에서 오는 증가된 키의 길이를 사용하는 경우보다 키의 길이를 증가시키지 않고, 스마트 카드와 같은 제한된 메모리에서 실용적으로 사용할 수 있게 하며, 암·복호화를 수행하는 처리 시간을 단축시킬 수 있고 키 관리 면에서도 여러 개의 공개키/비밀키를 사용하는 경우보다 실용적이 된다.

2. 로지스틱 함수

혼돈이론에서 가장 단순한 역학 체계 중의 하나는 로지스틱 함수(Logistic function)이다[3,6]. 혼돈이론(Chaos)은 외관상 복잡하고 비예측적이나 불규칙성의 이면에 공존하는 잘 정의된 질서구조를 밝혀, 다양한 분야에서 적극적인 연구 및 응용을 할 수 있는 현상이다. 지금까지의 공학에서는 대부분이 선형시스템의 모습을 나타내고 있지만 자연계의 현상과 같은 모델에서 비선형시스템으로 문제해결을 접근하고 있다. 카오스 개념이 정립되기 전에는 '측정이 불가능한 현상'은 잡음으로 처리해왔다. 잡음은 뜻을 알 수 없는 복잡한 성분으로 간주되었지만, 그 안에 내포된 현상에서 잘 정의된 질서구조와 단순한 행동에 따라 움직이는 경우도 있다. 이러한 비선형 시스템에서 가장 일반적으로 일어나는 현상이 카오스다. 카오스는 현재 자연과학 뿐만 아니라 공학, 의학, 사회과학 등 많은 분야에 파급되어 영향을 미치고 있다[4].

공학적인 관점에서 볼 때 카오스이론은 결정론적 비선형 동역학 시스템(Deterministic Non-Linear Dynamical System)에서 보편적으로 존재하는 복잡하고 예측 불가능한 잡음과 같은 현상으로 주기성이 없다. 카오스 연구는 19세기말부터 반세기 이상 명맥만 꾸준히 이어져 오다가 1975년 라이와 요크스는 "Period Three Implies Chaos"란 논문에서 카오스는 '결정론적 비선형 동역학 시스템에서의 복잡한 현상'이라고 정의하면서 과학기술 용어로 사용되기 시작하였다.

카오스 신호 생성부는 일반적으로 로지스틱 사상이나 간헐 카오스 사상과 같은 간단한 원리에 기초하여 카오스성을 갖는 시계열 신호를 생성한다. 따라서 본 논문에서는 다음과 같은 로지스틱 방정식[5]을 사용하여 카오스성의 시계열 신호를 생성한다.

$$F(x_{n+1}) = c\, x_n(1-x_n)$$

위의 로지스틱 방정식에서 x_n에서 x_{n+1}로의 변화를 로지스틱 사상(Logistic map) 이라고 한다. 이는 가장 단순한 역학체계 중의 하나이지만 온갖 종류의 복잡한 역학 현상을 표현할 수 있다. 이러한 로지스틱 함수 이외에도

$$x_{n+1} = c\,x_n(1-x_{n-1}), \qquad xe^{r(1-x)}, \qquad x\,[1+r(1-x)], \qquad \lambda x/(1+ax^b)$$

같은 많은 로지스틱 함수[5]가 있지만, 본 카오스 계산 모듈에서는 처음에 설명한 개체수 변화를 모델화[6]한 $F(x_{n+1}) = c\,x_n(1-x_n)$을 사용하였다. 로지스틱 함수는 초기조건에 따라 민감한 반응을 보인다. 이에 대한 결과값이 궤도적인 혼돈 상태를 보이기 위해서는 약 0.51~0.9 사이의 초기조건 범위를 입력해야 한다.

[표-2] 초기 조건의 변화에 따른 결과값

Iterate	x_0 = 0.5	x_0 = 0.51	x_0 = 0.749	x_0 = 0.8
1	1	0.999	0.752	0.640
⋮	⋮	⋮	⋮	⋮
8	0	0.301	0.466	0.402
9	0	0.842	0.995	0.962
10	0	0.530	0.018	0.148
11	0	0.996	0.071	0.504
12	0	0.014	0.262	1.000
13	0	0.058	0.774	0.000
14	0	0.219	0.699	0.001
15	0	0.686	0.841	0.004
16	0	0.861	0.534	0.015
17	0	0.477	0.995	0.059
18	0	0.998	0.017	0.222
19	0	0.007	0.073	0.690
⋮	⋮	⋮	⋮	⋮

[표-2]는 c=4.0으로 고정하고, 초기 조건을 다르게 입력함에 따라 Logistic 함수 결과값들의 변화된 반응을 볼 수 있다[5].

3. 암호화 알고리즘

컴퓨터에 저장되었거나 통신망을 통해 전송중인 데이터의 보호를 위하여 많은 방법들이 이용된다. 데이터나 시스템으로 물리적인 접근을 통제, 비밀번호의 일시적인 이용, 불법행위를 탐지·추적하는 침입탐지 기능, 운영체계의 강화 등 많은 수단이 있을 수 있다. 그러나 무엇보다 안전한 방식은 저장된 데이터나 통신망의 데이터에 대해 강력한 암호기술과 긴 길이의 키를 사용해 암호화하는 직접적인 데이터의 보호가 가장 효과적인 방책이다.

현대 암호는 크게 '암호 알고리즘' 과 '프로토콜'의 두 가지 측면으로 분류할 수 있다. 암호 알고리즘은 일반적으로 치환(substitution)과 전치(transposition)에 기초적인 원리를 두고 있으며[7], 블록 암호 알고리즘(block cipher), 스트림 암호 알고리즘(stream cipher), 공개키 암호 알고리즘, 확률론적 공개키 암호 알고리즘이 있으며, 프로토콜은 인증, 디지털 서명, 키 분배, 키 위탁/복구, 비밀분산, 전자화폐, 전자투표 등의 문제에 중점을 두고 있다.

암·복호화 키는 송·수신자간에 설정된 안전한 통신채널을 통해 동기화 시킨다. 대칭키 방식은 [표-3]에서 알 수 있듯이 인증문제, 키분배 문제뿐만 아니라 공개키 방식보다 관리해야 될 키의 수가 많고 인증, 전자서명의 구현이 용이하지 못한 개선점을 수반하고 있지만, 공개키 암호 알고리즘보다 키의 길이가 작으므로 암호화와 복호화를 연산하는 시간이 감소되는 이점을 가지고 있다.

[표-3] 대칭키&비대칭키 암호알고리즘의 비교

특성	대칭형 암호(비밀키)	비대칭형 암호(공개키)
암/복호화 키	동일	비동일
암호키	비밀	공개
복호키	비밀	비밀
관리 대상인 키의 수	많다	적다
키의 전송	필요	불필요
인증,서명 구현	곤란	용이
암/복호 속도	빠름	느림

공개키 암호 알고리즘은 비대칭형 암호 알고리즘으로도 불리는데, 수신자가 생성한 서로 다른 암/복호화 키를 갖는다. 수신자는 송신자가 해당 정보를 암호화할 수 있도록 암호키(공개키)를 공개하고, 복호키(비밀키)는 자신만이 철저히 관리한다. 따라서 대칭형(비밀키) 암호 알고리즘처럼 키의 분배 문제를 송·수신간에 미리 인지하고 있던지 또는 키 분배센터(KDC)를 따로 두는 번거로움을 해소할 수 있다. 이처럼 비밀키 암호 알고리즘에서 다루기 어려웠던 몇몇의 문제들을 해결하고자 공개키 암호 알고리즘이 출현하게 되었다.

대표적인 사례로는 DH, RSA, Rabin, ElGamal[10], ECC[11], Knapsack, LUC, McEliece, LRP, Graph 등이 있지만, 본 논문에서는 실제적인 메시지를 암호화하기 위해 일회성[8,9]을 내포하는 비밀키, 공개키를 생성하고 이를 RSA 암호 알고리즘에 적용하여 본다.

공개키 암호알고리즘이 주로 사용되는 부분은 두 가지로 볼 수 있는데, 하나는 비밀키 암호 알고리즘처럼 특정 메시지를 암호화하는 경우와 전자서명에 사용되

는 경우다. 공개키 암호 알고리즘은 비밀키 암호 알고리즘보다 키의 길이가 길어서 대용량의 메시지를 암/복호화하는 경우에는 수행시간이 오래 걸리므로 일반적으로 암호화하는데만 사용되며, 전자서명의 경우처럼 제한된 크기의 해쉬 결과와 같은 메시지를 처리하는데 사용된다.

또한 공개키 암호알고리즘은 소인수 분해문제(Factorization problem)와 이산대수 문제(Discrete logarithm problem), 평방 잉여 문제(Quadratic residuosity problem)[12]등으로 구성되는데 본 논문은 암호방식 중에서 소인수 분해를 기초로 한 RSA만을 살펴보도록 한다.

1977년 Rivest, Shamir, Adleman에 의해 개발되고, 1978년 공개된 RSA 공개키 암호알고리즘[1,2]은 암호화와 전자서명 등을 제공할 수 있으며 안전도의 근간을 소인수 분해의 어려움에 두고 있다. DES에 비해 S/W로 구현했을 때 100배 정도, H/W로 구현했을 때 1000배 ~ 10,000배정도 느리기 때문에 일반적으로 RSA 암호 알고리즘은 비밀키 암호 알고리즘과 함께 사용된다.

이와 같은 RSA 암호알고리즘은 Euler의 정리를 사용하는데, 양의 정수의 집합인 {1, 2, …, n-1}의 원소들 중에서 n과 서로소의 관계에 있는 원소들의 개수를 $\varphi(n)$으로 나타내며, 이를 Euler의 φ 함수라 한다. p를 소수라고 할 때 서로소인 원소들의 개수 $\varphi(n) = p-1$개가 된다. 여기서 n이 두 소수 p와 q의 곱일 때 $\varphi(n) = (p-1)(q-1)$이며, 소인수 분해 없이 n을 구하기란 어려운 문제가 된다는 이론적 기준에서 알고리즘이 설계되었다.

4. 키 생성 모델의 설계 및 구현

4.1 개발환경

일회성 비대칭키 구현을 위한 시스템 구성 환경은 [표-4]와 같다.

[표-4] 일회성 비대칭키 개발환경

개발환경	Application
시스템	SUN SPARC 1
운영체제	SunOS 5.5.1
컴파일러	cc 4.0 gcc 2.7.6.1
알고리즘	MD5, Chaos (Logistic Func.) RSA (RSAEURO Beta Release 1.03)
결과값 측정	Mathematica 4.0

4.2 키 생성 모델

일회성 비대칭키를 생성하는 과정을 크게 5가지로 분류하였다. 첫째 PIN(Personal identification Number)을 입력하는 단계와, 둘째 사용자 식별정보를 간략화(digest) 시키는 단계, 세 번째 임의적인 두 개의 chaos 결과값을 추출하는 단계이며, 네 번째 공개키와 비밀키를 생성하는 단계이고, 마지막 과정으로 데이터를 암호화(encryption), 복호화(decryption)하는 단계이다. 이러한 각각의 과정을 좀 더 상세히 살펴보도록 하겠다.

4.2.1 PIN 입력단계

사용자 고유의 식별정보(PIN)를 입력하는 세 번째 단계로 카오스 결과값을 추출하기 위해 필요한 초기값을 입력하는 과정이다.

[표-5]는 이러한 입력단계의 프로토타입과 설명이다

[표-5] 사용자 정보 입력 단계의 프로토타입

프로토타입	설 명
int controlsock(sock) int sock;	Berkeley IPC에서의 메시지 송·수신을 담당하는 함수로써 암호화된 메시지를 수신할 실체로부터 PIN 정보를 입력받고 해당 정보를 Digest 단계로 넘겨준다.

4.2.2 PIN 정보 Digest 단계

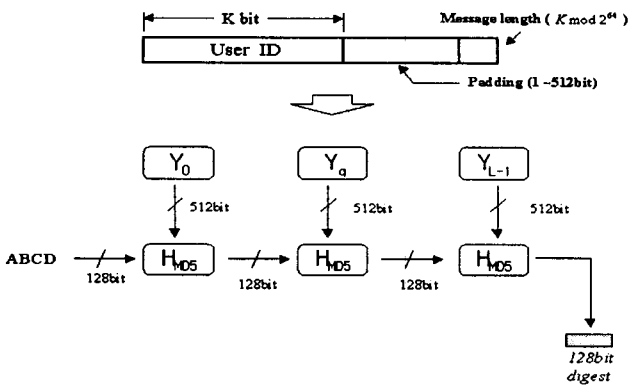
본 단계에서는 임의의 가변적인 길이인 사용자 정보를 고정적인 길이의 메시지로 변환하는 단계로써 MD5 해쉬함수[13,14]를 사용하여 Digest 시킨다. 이 모델에서 해쉬함수를 적용시킨 이유는 두 가지로 요약할 수 있다.

먼저 상태공간에서의 규칙(Dynamics)이 사전에 결정되어져 있기 때문에 초기조건이 확정됨과 동시에, 그에 따른 궤도가 유일하게 결정되어진다. 초기조건에 대한 궤도적인 특성을 달리하는 카오스 함수의 입력값으로 사용하기 위해서는 가변적인 길이를 갖는 사용자 식별 번호를 카오스 상태를 보이는 로지스틱 함수의 결과값을 유발할 수 있는 고정적인 길이의 초기조건 범위로 Digest 시켜야

한다.

두 번째로 불법적인 실체가 소인수 분해 문제를 해결하려는 공격 형태와는 또 다른 형태로 공격해 올 경우를 방어해야 한다. 가령, 공격자가 공개키를 이용하여 비밀키를 찾기 위해서는 소인수 분해 문제가 필수적으로 수반되는데 소인수 분해를 행하지 않고 직접 사용자 식별 번호를 찾는 역계산을 시도할 수도 있을 것이다. 이런 경우 해쉬함수와 카오스 함수는 역계산의 복잡도를 증가 시키는 이득을 준다. 해쉬함수 자체가 역계산이 불가능하도록 고안되었으며, 카오스 함수는 사용하는 로지스틱 함수마다 카오스적인 상태를 보이는 초기조건과 반복계수에 대한 모든 특성과 분포를 인지하고 있어야 되는 난해함을 제공하여 준다.

이러한 두 가지 기능을 제공하기 위해 해쉬함수의 사용은 불가피하며, [그림-1]은 Digest하는 모델의 구성도[2]를 보여주고 있다.



[그림-1] 사용자 정보 Digest 과정

[그림-1]의 처리 과정은 먼저, 입력받은 사용자 정보 메시지(K bit)가 512비트의 배수가 되도록 zero비트를 추가하여 패딩 과정을 거친다. 이 패딩 과정에서는 512비트의 블록으로 분할하는데 이 512비트 중 64비트는 메시지의 길이에 대한 정보로 이용됨으로 512 - 64 = 448비트의 배수로 패딩 비트를 추가하면 된다.

A = 01234567 B = 89ABCDEF C = FEDCBA98 D = 76543210

위와 같이 해쉬함수의 중간 및 최종 결과값을 저장하기 위해 32비트 버퍼인 A, B, C, D는 Hexa 값으로 초기화된다. 다음은, 패딩 과정을 거쳐 512비트 블록으로 분할된 메시지와 4개의 버퍼값 (128비트)을 입력값으로 하여 기약 논리 함수를 사용한 4단계의 라운드, 64 연산과정을 실행하면서 버퍼값을 갱신한다. 각각의 라운드는 비슷한 구조를 가지고 있지만 서로 다른 기약 함수를 가지며 이 기

약 함수는 3개의 32비트 워드를 입력으로 하여 1개의 word를 출력한다.

[표-6] 사용자 정보 Digest 단계의 프로토타입

프로토타입	설 명
void MD5Init (mdContext) MD5_CTX mdContext;	MD5 해쉬함수에 사용되는 4개의 버퍼를 초기화 한다.
void MD5Update (mdContext, inBuf, inLen) MD5_CTX mdContext; unsigned char *inBuf; unsigned int inLen;	메시지 길이(K mod 2^{64})와 입력받은 PIN 정보를 512비트 블록에 할당한다.
void MD5Final (mdContext) MD5_CTX mdContext;	패딩 비트를 부가하고 최종적인 4개의 버퍼값 128비트 Digest 메시지를 생성한다.
static void Transform (buf, in) unsigned long int *buf; unsigned long int *in;	4개의 버퍼값 128비트와 512비트 블록을 입력으로 받아 기약 논리 함수를 사용한 4단계의 라운드, 64 연산과정을 실행한다.

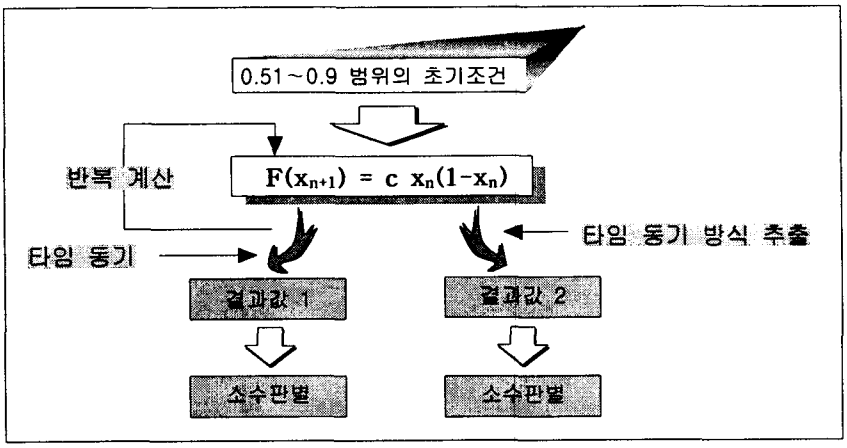
이와 같이 512비트 블록과 128비트 버퍼값을 입력으로 하여 128비트의 결과값을 출력하는 과정을 L-1번 실행한 후 L번째의 실행 결과값이 Digest된 128비트 메시지가 된다. 이러한 과정의 프로토타입은 [표-6]에 따른다.

4.2.3 카오스 결과값 추출 단계

이 단계는 3가지의 부분적인 과정으로 분류할 수 있는데 먼저, 카오스의 결과값이 혼돈 상태가 되기 위해 digest된 128비트 메시지를 초기값 영역으로 변환하는 단계가 있다. 다음 단계는 카오스의 로지스틱 함수를 이용하여 반복 계산한 결과값들 중에서 임의의 결과값을 선택하는 단계이며, 마지막 단계는 선택된 카오스 결과값을 바탕으로 소수를 판별하는 단계이다. 이러한 3가지 과정을 좀더 자세히 살펴보기로 하겠다.

첫 번째 과정은 전단계에서 Digest된 128비트 메시지를 카오스 초기조건의 범위로 변환하는 과정이다. Digest된 128비트 메시지 중 타임 동기 방식을 사용하여 임의의 32비트를 추출한다. 이 과정에서 불법적인 실체가 카오스의 초기값을 운 좋게 획득했을지라도 불완전한 32비트 값이므로 나머지 96비트를 추가적으로 알아내야하는 어려움과 32비트의 위치를 알아내야하는 난해함이 존재한다. 다음 절에서 설명할 로지스틱 함수는 초기조건에 따라 민감한 반응을 보인다. 추출된 32 비트는 8문자의 Hexa값으로 변환되어 로지스틱 함수의 결과값이 궤도적인 카오스 상태를 보이도록 약 0.51~0.9 사이의 초기치 범위로 변환된다.

다음 과정은 초기조건의 범위로 변환된 값을 로지스틱 함수의 x_0 조건으로 입력하여 타임 동기 방식에 따른 2개의 결과값을 반복 계산하여 얻는다. [그림-2]는 이러한 처리 절차를 보여주고 있다.



[그림-2] 카오스 계산 처리절차

마지막 과정에서는 로지스틱 함수의 반복 계산 중에 타임 동기와 일치하는 x_n 값을 선택하여 소수판별 과정의 입력값으로 할당하고, 소수판별 과정을 거쳐서 2개의 일회성 소수를 생성하게 된다.

본 논문에서 카오스 함수로 로지스틱 함수를 적용시킨 목적은 궤도적인 혼돈상태를 보이는 결과값을 바탕으로 보다 안정적인 일회성 소수를 생성함이 목적이다. 비밀키는 암호화된 메시지를 복호화 시키려는 실체가 가지고 있는데, 그러한 키를 생성하기 위해서는 복호화 하려는 실체가 2개의 큰 소수를 선택하는 문제를 수반한다. 따라서 궤도적인 혼돈상태를 보이는 시계열을 바탕으로 보다 안정적인 소수를 선택하여 일회성의 성질을 얻기 위해 카오스 함수를 적용시켰다.

소수의 정의는 1과 자기자신 이외에는 나누어 떨어지는 정수가 없는 양의 정수를 말한다. 임의의 큰 수가 소수인지 아닌지를 결정하는 간단하고 효율적인 방법은 없다. 단지 어떤 소수판별 방법이 시스템의 환경에 적합한가를 판단하는 것이 우선책이 될 것이다. 소수 판별 방법에는 Solovay-strassen, Miller-Rabin 방법[1,2]등이 있고, 직접 소수를 구하는 방법에는 에라토스테네스의 체(Eratosthenes' sieve)[2]등 여러 가지 방법이 있다. 본 과정에서는 소수를 판별하고 소수가 아니면 가장 가까운 소수를 직접 찾아내는 방법을 사용한다. 따라서 가장 기본적인 소수판별 방법 중에 하나로써 알고리즘은 다음과 같다.

```
LET  정수 ← CNT, C, SQRN
CNT  ← 2
SQRN  ← (int) SQRT(C)
FOR  CNT ← 2 upto CNT < SQRN
     IF C%CNT++ == 0
          C = C-1
          CNT  ← 2
          RETURN !TRUE
     ENDIF

RETURN C
ENDFOR
```

[표-7] 카오스 결과값 추출단계의 프로토타입

프로토타입	설 명
void MDPrint (mdContext) MD5_CTX mdContext;	128비트의 Digest된 메시지 중 임의의 32비트를 추출하여 카오스 함수의 초기조건 범위로 변환한다. 카오스 결과값들 중에서 타임 동기 방식을 사용하여 2개의 결과값을 선택한다.
int mk_prime (n) int n;	카오스 함수의 결과값 중 선택된 2개의 결과값이 소수인지 아닌지를 판별하고 아니라면 결과값보다 작고 가장 가까운 소수를 찾아 return 해준다.

소수 판별을 행하려는 대상값 C를 입력받아 2부터 C의 제곱근까지 소수 판별을 행한다. 만약 C % CNT가 '0'이 되면 즉, 소수가 아니면 C보다 작은, 가장 가까운 소수를 찾아 반환한다. 이 과정에서 유의할 점은, IF문의 조건을 만족하는 경우에는 반드시 C를 하나 감소시키고, 제수 CNT의 증가를 초기치 2로 되돌린 후에 적용시킴으로써 C가 소수가 아니더라도 C에서 가장 가까운 소수를 찾게 된다는 점이다. IF문의 조건에 만족하지 않는 경우에는 C가 바로 소수이므로 해당하는 값을 반환할 것이다.

[표-7]는 카오스 결과값 추출단계의 부분적인 3가지 과정의 프로토타입과 설명을 보여주고 있다.

4.2.4 공개키 · 비밀키 생성단계

본 과정에서는 2개의 소수 p, q를 바탕으로 공개키{n,e} · 비밀키{n,d}를 생성하는 과정을 설명한다. 전 과정에서의 p, q를 바탕으로

$d = e^{-1} \bmod[(p-1)(q-1)]$ 식으로부터 e와 d의 값을 유도해낸다. 이러한 과정에서는 몇 가지 유의해야 할 사항이 있다. 첫째, '두 개의 큰 소수 p와 q를 어떤 방식으로 결정하는가'의 문제이다. 가령 이 문제를 해결하기 위해 적당히 큰 수를 선택하여 확률론 적인 방법과 같은 소수 판별 방법을 통해 소수인지 아닌지를 판별할 수도 있을 것이다. 둘째, e와 d 중에서 하나는 선택하고 다른 하나는 계산되어져야 한다. 셋째, e를 선택할 경우 φ(n)과 서로소인 임의의 정수이어야 한다.

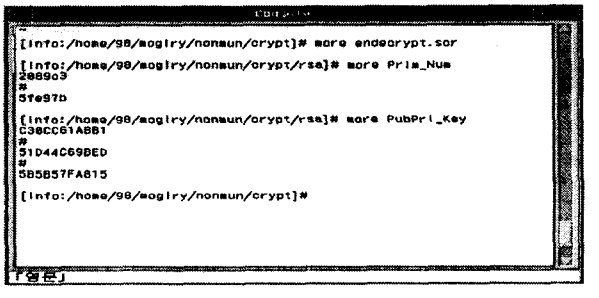
우선, 1부터 φ(n)까지의 정수를 CNT라고 했을 때 φ(n) % CNT의 결과가 '0'이면 CNT는 φ(n)의 약수이다. 이런 φ(n)의 약수의 집합을 DIV라고 했을 경우, 1부터 φ(n)까지의 정수에서 DIV의 원소와 동일한 정수를 제외한 나머지를 TMP라고 하자. 그렇다면 φ(n)과 서로소인 e는 TMP 원소들 중 하나이다. TMP의 원소들 중에서 랜덤하게 하나를 선택하여 e에 할당한다. e의 약수의 집합을 E_DIV라 한다면 E_DIV[0] 즉, 1을 제외한 E_DIV[1], E_DIV[2], E_DIV[1],…… 와 같은 원소들과 DIV[1], DIV[2],…… 등을 비교하여 동일한 인수가 존재한다면 랜덤하게 선택된 e는 φ(n)과 서로소가 아니다. 왜냐하면, DIV[0]와 E_DIV[0]의 값인 '1'이외에는 공통의 인수가 존재해서는 안되기 때문이다. 따라서 동일한 인수가 존재하면 e의 값을 랜덤하게 다시 갱신해야 한다. 만약 동일한 인수가 '1'이외에는 존재하지 않는다면 e는 φ(n)과 서로소인 관계를 갖게 된다. 이처럼 φ(n)과 서로소인 e를 정하는 문제는 다음과 같다.

```
LET  Q ← φ(n)
DIV ← Q의 약수들의 집합
TMP ← 1~Q의 정수 중에서 DIV를 제외한 집합
FOR  CNT ← 1 upto CNT <= Q
   TMP++ ← CNT;
   IF Q%CNT == 0
      DIV++ ← CNT
```

```
FOR  CNT ← 1 upto CNT <= Q
  IF TMP[CNT]==DIV[1] || TMP[CNT]==DIV[2] |  TMP[CNT]==DIV[3]||·····
     WHILE( !TMP[NULL])
        TMP[CNT] ← TMP[CNT+1]
        CNT ← CNT+1
NUM_LEN ← strlen(TMP)
RND ← RAND % NUM_LEN
E ← TMP[RND]
FOR  CNT ← 1 upto CNT <= E
  IF E%CNT == 0    E_DIV++ ← CNT
FOR  J ← 1 upto CNT <= NUM_LEN
  IF DIV[J]==E_DIV[1] || DIV[J]==E_DIV[2] ||
                         DIV[J]==E_DIV[3]||·····
```

선택된 e를 바탕으로 d를 계산하는 과정은 다음과 같다.

```
TMP ← 1부터 φ(n)까지의 정수에서 φ(n)의 약수를 제외한 원소들의 집합.
FOR CNT ← 1 upto CNT <= TMP 원소 수
IF (e * TMP[CNT]) % φ(n) == 1
   d ← TMP의 해당 원소
```



[그림-3] 일회성 공개키/비밀키 생성

위와 같은 방법으로 일회성 소수로부터 공개키 n, e와 비밀키 n, d를 구할 수 있다. 아래의 [그림-3]은 두 개의 소수 p, q를 가지고 n, e, d 키를 생성하는 과정을 보여주고 있다.

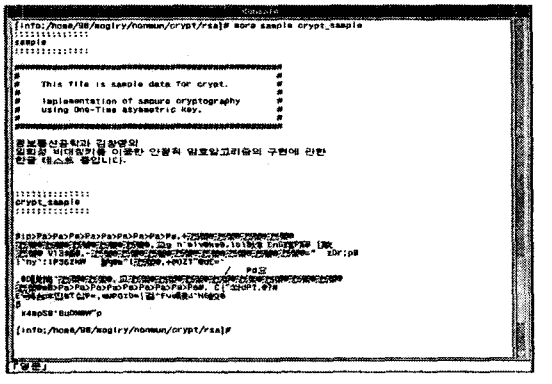
4.2.5 암호화 · 복호화 단계

RSA 공개키 알고리즘을 전자서명과 같은 응용에 사용한다면, 자신의 비밀키로 암호화하고 공개키로 복호화를 행하여 서명을 확인하기도 하지만 일반적인 메시지 암호화에서는 공개키로 암호화하고 비밀키로 복호화를 행하는 것이 속도가 빠르다. 그러나 대칭형 암호알고리즘보다 긴 길이의 키를 사용하므로 수행 속도가 현저히 느리다.

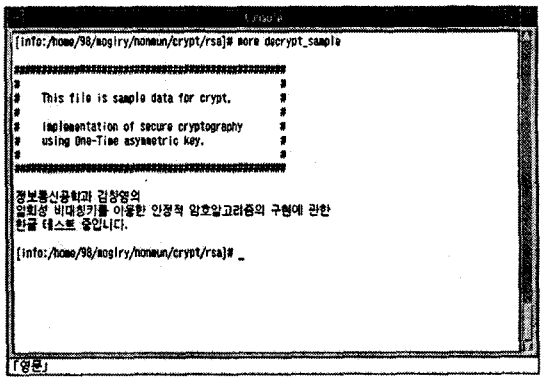
암호화 라이브러리에는 대표적으로 RSAREF와 RSAEURO, SSLeay 3가지 종류가 있다. RSAREF와 RSAEURO는 거의 동일한 인터페이스를 제공하는데, RSAREF는 미국의 RSA사가 public domain으로 RSA, DES, MD5 알고리즘 등을 C코드로 구현하여 공개한 라이브러리이다. API가 간단하고 명료하여 암호화 시스템을 개발하는데 적용하기 쉽지만 RSA 사로부터 라이센스를 얻어 사용한다. RSAEURO는 유럽, 영국에서 개발된 암호화 라이브러리이기 때문에 미국의 수출 금지법에 적용을 받지 않는다.

마지막으로 SSLeay는 호주의 Eric Young에 의해 만들어진 라이브러리이다. SSL 프로토콜을 위해 만든 패키지로서 SSL API는 비교적 명확하고 사용하기에 용이하게 되어있지만 다른 암호화 알고리즘에 관련된 API는 이해하기가 어렵게 되어있다. 본 과정에서는 RSAEURO Beta Release 1.03을 사용하여 암/복호화에 적용한다. 암/복호화에 적용되는 기본적인 함수는 RSAPublicEncrypt(), RSAPublicDecrypt(), RSAPrivateEncrypt(), RSAPrivateDecrypt() 4가지 형태로 제공되지만, 공개키로 암호화를 수행하고 비밀키로 복호화를 수행하기 위해 RSAPublicEncrypt() 함수와 RSAPrivateDecrypt() 함수만을 사용하였다.

$$
\begin{aligned}
&3^2 \pmod{11} = 9 \\
&3^4 \pmod{11} = 9^2 \pmod{11} = 81 \pmod{11} = 4 \\
&3^8 \pmod{11} = 4^2 \pmod{11} = 5 \\
&3^{16} \pmod{11} = 5^2 \pmod{11} = 3 \\
&3^{32} \pmod{11} = 3^2 \pmod{11} = 9
\end{aligned}
$$



[그림-4] 일회성 키에 의한 암호화 결과



[그림-5] 일회성 키에 의한 복호화 결과

이러한 함수의 암·복호화 과정에서는 모듈러 연산을 사용하는데 모듈러 연산의 특징 중에 하나는, ***[(a×b) mod n]=[(a mod n)×(b mod n)] mod n*** 과 같은 계산이 가능하다. a, b가 대단히 큰 수 이었다면 a×b 값의 크기 또한 큰 수가되는데 큰 수인만큼 mod n으로 축소하는 단계도 많아진다. 가령 3^{32}(mod 11)을 구하려고 한다면 3을 32번 곱한 후 mod 11로 축소할 필요 없이, 3^2, 3^4, 3^8, 3^{16}, 3^{32} 와 같이 5 번의 곱셈만을 계산하면 된다.

RSAEURO라는 라이브러리와 일회성 공개키 {n, e}, 비밀키 {n, d}를 사용하여 암호화, 복호화과정에 적용하였다. [그림-4]은 일회성 공개키를 바탕으로 소스 파일을 암호화한 내용을 보여주고 있으며, [그림-5]는 일회성 비밀키를 바탕으로 암호화된 파일을 다시 복원한 내용을 보여주고 있다.

5. 실험 및 고찰

본 절에서는 이제까지 제안하고 구현된 요소들을 바탕으로 몇 가지 결과값을 측정하여 본다. 첫째, 동일한 사용자 식별 정보를 입력했을 경우의 n, e, d 각각의 키에 대한 결과값을 살펴보고 둘째, 사용자 식별 정보를 입력한 시기부터 일회성 키가 생성되는 시점까지의 처리 시간과 CPU time을 알아보도록 하겠다. 마지막으로 일회성 키를 사용하여 암호화 및 복호화를 수행하는 처리속도와 CPU time을 살펴본다.

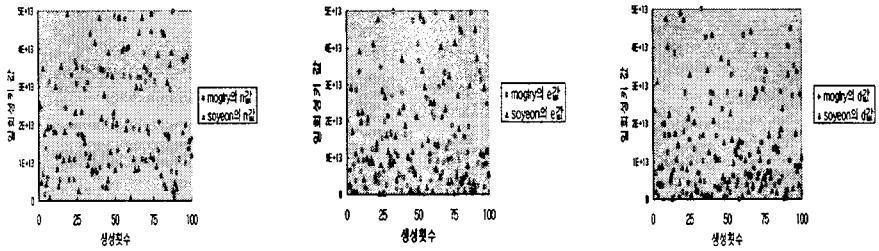
5.1 동일한 PIN에 대한 일회성키 측정

동일한 사용자 식별 정보의 입력에 대하여 일회성을 갖는 n, e, d 3개의 키값과 그에 대한 분포를 살펴보고, 두 명의 PIN에 대한 일회성 키값을 비교하여 본다.

생성횟수 / PIN		1	2	3	...	54	55	56	...	98	99	100
moglry	n	465715A08115	15FA273F03A3	6653C1C7DEF		87D6E64B839	4F89D3D87099	1DEF72C7B2E5		C68DEC9A99B	2E7E0AABE36D	EC928574A15
	e	40DF2A9ECE1D	60514B98953	1CB1390419D	...	18D3474E61	3FC45E5D33F9	17FD5D5C1F95	...	79150D07B5F	1668BBBD5A93	5150D6B7B95
	d	3B4B2006D92D	61E1DF473DB	3A4718A9F75		5ABF49B2DA1	3C8FEADE3939	34C9C29BFBD		79F8901A8FF	190206AFA137	169DDA75705
soyeon	n	17258F8F854B	47785C72ED3	1FFB6A7C4929		49361427FC67	2404560E64B3	249475DEFBBB		E5B40395B33	429B243EA2B1	AF664CFF275
	e	217E3238369	28BF13AC0A5	1A9421D0AB81	...	2B43203D4FD7	1AB13C64237B	1C2BE592ED5	...	AD84A9C5A5B	151C319F4AE7	4F0630A9EC9
	d	15A7196 43699	1E37EC641ED	1C651A2B1081		8E207154147	BB79CB77E33	AE0BCFA447D		3AD0D3ED943	83D01F8988F	A15DB41EF7D

[표-8] 사용자 정보별 일회성 키

[표-8]은 PIN 정보에 따라 생성된 각각의 n, e, d 키값의 Hexa값 분포를 나타내며 [그림-6]에서는 이를 Decimal로 변환하여 분포를 비교한다. [그림-6]의 x축은 일회성키 n의 생성횟수를 나타내며, 0~5E+13 까지를 y축으로 할당하였다. 그림에서 알 수 있듯이 y축의 한 구간은 1E+13인데 n값의 분포보다 e, d의 분포가 아래쪽으로 치우친 현상을 볼 수 있다.



[그림-6] 일회성 키값별 분포도

이러한 현상은 일회성 소수를 찾을 때 2개의 카오스 결과값보다 작은 소수를 찾아내거나, RSA 공개키 알고리즘에서 e와 d를 선택·계산할 때 n보다 작은 서로소인 수를 사용함으로 n값보다는 e나 d값이 작은 쪽에 위치하게 된다.

하나의 사용자 식별 정보로부터 생성되는 일회성 키는 n, e, d가 있는데 여기에서 공개키로 사용하는 것은 n과 e이며 비밀키로 사용하는 것은 n과 d이다. 물론 반대로 공개키{n,d}, 비밀키{n,e}를 사용할 수도 있겠지만 앞에서 설명한바와 같이 d와 e중 하나는 $\varphi(n)$과 서로소인 수를 선택해야 하며 나머지 하나는 선택된 수를 바탕으로 계산되어져야 한다. 본 논문에서는 e값을 먼저 선택하고 d값을 계산하였기 때문에 n과 e를 공개키로 n과 d를 비밀키로 사용하였다.

5.2 일회성키 생성시간 측정

본 절에서는 사용자 식별 정보를 입력하는 시기부터 n, e, d의 일회성 키가 생성되는 시간까지의 생성시간을 달력시간과 프로세스 시간으로 나누어 각각을 측정하였다. 이러한 측정결과는 [표-9]에서 볼 수 있다. [표-9]와 같은 5개의 일회성키 생성시간은 SUN SPARC 1시스템의 SunOS 5.5.1에서 측정한 결과로써 단위는 초(sec)이며, CPU time은 user time을 말한다. 약 50비트의 적은 길이의 키를 생성함으로 키의 생성시간이 1024비트의 키를 사용하는 경우보다 짧은 것은 당연하다.

[표-9] 일회성 키의 생성시간 (단위 : sec)

키 생성시간 / 횟수별 일회성키			키 생성 시간	
			Calendar time	CPU time
1	n	2B550CF94A6F	0.16000	0.02000
	e	D0F167FC6C5		
	d	DD4310DE1DD		
2	n	1703990F719D	0.14000	0.03000
	e	285DB28F6A5		
	d	F66C3B2E6D		
3	n	EA9BFF7A25	0.15000	0.05000
	e	3EB4A9D585		
	d	AD2A8BF24D		
4	n	48CF3896C3B3	0.14000	0.04000
	e	45A39179AB05		
	d	2D47874617D5		
5	n	51EF3C47C62D	0.13000	0.02000
	e	4222336B9371		
	d	174BC2A1DE11		

[표-10] 암 · 복호화 처리시간 (단위:sec, 파일:2068 bytes)

처리시간 / 일회성키			암 · 복호화 처리시간	
			Calendar time	CPU time
n	2B550CF94A6F	Encryption	6.520	6.360
e	D0F167FC6C5			
d	DD4310DE1DD	Decryption	6.560	6.240
n	1703990F719D	Encryption	5.970	5.820
e	285DB28F6A5			
d	F66C3B2E6D	Decryption	6.060	5.830
n	EA9BFF7A25	Encryption	5.980	5.820
e	3EB4A9D585			
d	AD2A8BF24D	Decryption	6.250	6.030
n	48CF3896C3B3	Encryption	6.650	6.520
e	45A39179AB05			
d	2D47874617D5	Decryption	6.760	6.590
n	51EF3C47C62D	Encryption	6.520	6.390
e	4222336B9371			
d	174BC2A1DE11	Decryption	6.460	6.310

5.3 암호화/복호화 시간

5.2절에서 생성된 5개의 일회성키 n, e, d를 토대로 간단한 예제 파일을 암호화하는 시간과 복호화하는 시간을 측정하였다. 5.2절과 마찬가지로 암·복호화 처리시간은 달력 시간과 프로세스 시간으로 나누어 각각을 측정하였다. 이에 대한 측정결과를 [표-10]에서 볼 수 있다. [표-10]은 [표-9]에서 생성된 일회성키 n, e, d를 이용하여 암·복호화를 실행하는데 걸린 달력 시간과 프로세스 시간을 보여주고 있다. 파일 크기가 2068 Byte인 파일에 대한 처리시간에서 calendar time은 최소 5.970 sec에서 최대 6.760 sec가 소요되는데 시스템 자원의 사용률에 따른 약간의 시간 차이를 볼 수 있었다. 더불어 본 논문의 구현 시에 사용된 파일 입·출력 시스템 콜 등의 사용 때문에 약간의 처리시간이 증가하는 요인이 되었다.

6. 결론

암·복호화 연산시간은 키길이에 비례하기 때문에 키의 길이가 해를 거듭할수록 증가하게 되었다. 따라서 본 논문에서는 가변적인 길이의 사용자 식별 정보를 MD5 알고리즘을 토대로 고정 길이의 정보로 간략화 시켰다.

간략화된 정보는 로지스틱 함수의 결과값이 궤도적인 카오스 상태의 시계열을 유도하는 초기 조건의 영역으로 변환되고, 로지스틱 함수의 반복 계산 중에 타임 동기와 일치하는 2개의 결과값을 선택하여 소수판별을 거친 후 일회성을 갖는 소수를 생성하였다. 이러한 소수를 바탕으로 권장키보다 짧은 암·복호화 키를 생성할 수 있었고, 이를 RSA 알고리즘에 적용하여 암·복호화 과정을 수행할 수 있었다.

권장키보다 짧은 암·복호키를 적용한다면, 키 생성시간의 감소와 암·복호화 처리시간을 단축시킬 수 있을 뿐만 아니라 암호시스템에서 키를 실용적으로 관리할 수 있으며, 2개의 로지스틱 함수 결과값에서 소수를 판별하는 방법에 따라 상대적인 키 생성 시간과 일회성을 갖는 상이한 키값을 측정할 수 있었다. 따라서 효율적인 처리시간을 얻을 수 있는 개선된 소수판별방법을 사용한다면 키 생성시간의 단축과 더불어 짧은 길이의 암·복호키의 사용에 대한 신뢰성을 증대시킬 수 있음을 확인하였다.

이에 대한 활용방안으로 스마트 카드와 같은 제한된 메모리에 적용이 가능하므로 안정적인 키의 길이를 찾아내는 연구가 기대된다.

참고문헌

[1] Bruce Schneier, "Applied cryptography second edition : protocols, algorithm, and source code in c", John Wiley & Sons, Inc., 1996.
[2] william stallings, ph.D., Network and internetwork security principles and practice", Prentice-Hall, Inc., 1995.
[3] Roman E. Maeder, "Function Iteration and Chaos", Mathematica Journal, Vol 5., 1995.
[4] Denny Gulick, "Encounters with Chaos", McGraw-Hill, Inc., 1992.
[5] Robert L.Devaney, "Chaos, Fractals, and Dynamics ; computer experiments in mathematics", Addison-wesley, 1990.
[6] 정성용, "카오스 이론을 이용한 암호화기법", 한국정보과학회, Vol. 25. No.2, pp45-47, 1998.
[7] 김 철, "암호학의 이해", 영풍문고, 1996.
[8] Haller, N., "The S/KEY One-Time Password System", RFC 1760 February 1995.
[9] Haller, N., Metz, C., Nesser, P. and M. Straw, "A One-Time Password System", RFC 2289, February 1998.
[10] T.ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", IEEE Trans. Infer. Theory, July 1985.
[11] 이인수, "타원곡선 공개키 암호시스템현황", 한국정보보호센터, 1998. 4.
[12] "NETSEC-KR '99 ", 한국정보보호센터 & 한국통신정보보호학회, 1999.
[13] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.
[14] Federal Information processing standards publication 180-1, "Secure hash standard", 1995.
[15] Charles P. Pfleeger, "Security in Computing Second Edition", Prentice Hall PTR, 1997.
[16] Anup K. Ghosh, "E-commerce security", John Wiley & Sons, Inc., 1998.
[17] 김창영, 이 형, 홍성찬, "Chaos에 의한 비대칭형 다층 OTP 생성기법에 관한 연구", 한국정보처리학회, 제6권. 제1호, 1998. 4.

Implementation of Practical cryptography using one-time asymmetric key

Lee, Hyoung and Kim, chang-Young

In this paper, we have discussed five-steps generating one-time asymmetic key. The first step is to input the PIN(Personal Identification Number), the second step is to digest the PIN information, the third is to draw out the result value of experiment, the fourth is to generate the public and private key, and the last is data encryption and decrytion step.

As an improvement of primary decomposition algorithms and performance of computer systems, we should encrypt and decrypt with increased key length, we will be able to decrease the encryption and decryption processing time as well as system load.

◆ 저자소개 ◆

이 형(Hyoung Lee)

1964년 서울대학교 수학과 이학사
1971년 성균관대학교 전자계산학과 경제학석사.
1992년 조선대학교 컴퓨터 공학과 공학박사.
1993~현재 대전대학교 정보통신공학과 교수 및 공과대 학장
관심분야 : 컴퓨터 그래픽, 영상처리, 암호화

김창영(Chang-Young Kim)

1998년 우송대학교 컴퓨터과학과 졸업 (학사)
1998년~현재 대전대학교 정보통신공학과 대학원 입학
2000년 대전대학교 정보통신공학과 대학원 졸업예정
관심분야 : 영상처리 및 암호화